

# A BRIEF INTRODUCTION TO R

STAT203

YuehWen Liao

Dept. of Statistics, Stanford Univ.

# What is R



- A software & a dialect of the S language
- A system for statistical analysis and graphics
- Works in various OS.
- Free (available at <http://www.r-project.org/>)
- An interpreted language => easy to program.

# A simple example - "Hello, World!"



- Just type

```
> print("Hello, World!")
```

- Or

```
> x <- "Hello, World!"
```

```
> print(x)
```

# Getting Help



- To get the usage of a function
  - `help("lm")`
  - `help(lm)`
  - `?lm`
- To open HTML help
  - `help.start()`
- To search a function
  - `help.search("lm")`
- search on Google!

# Object

- Two intrinsic “properties”: mode and length
- Four modes:
  - numeric  
ex: `a=1, 1.2, 2.4e25, Inf, -Inf, NaN`
  - character  
ex: `char=“something”, “hello”, “2”`
  - complex  
ex: `c= 1+2i, 1.1+2.4e4i`
  - logical (FALSE or TRUE)
- Length: the number of elements of the object

# Object



- vector
- matrix
- array – a *k*-dimension “matrix”
- factor – a categorical variable
- data frame – a table composed with one or several vectors and/or factors all of the same length but possibly in different modes
- list – a list can contain any type of object
- ts–time series

# Vector

- Vector- must have their values all of the same mode
  - > vector("integer", 10) / integer(10) / c(1, 2, 3, 4)
  - > vector("logical", 10) / logical(10) / c(T, F)
  - > vector("numeric", 10) / numeric(10) / c(e^10, -0.98)
  - > vector("character", 10) / character(10) / c("a", "b")
- Common command for vector
  - min/max, sum, mean, var, sd
  - length
  - sort/sort.list
  - seq/rep

# Some examples of Vector

```
> seq(from=0,to=100,by=10) (or seq(0, 100, by=10) )
```

```
[1] 0 10 20 30 40 50 60 70 80 90 100
```

```
> seq(from=0,to=100,length=11)
```

```
[1] 0 10 20 30 40 50 60 70 80 90 100
```

```
> rep(c(1,2,3,4),time=c(1,2,3,4))
```

```
[1] 1 2 2 3 3 3 4 4 4 4
```

```
> rep(c(1,2,3,4),time=c(4,3,2,1))
```

```
[1] 1 1 1 1 2 2 2 3 3 4
```

```
➤ sort(c(3,1,5,8,-4)) / sort(c(3,1,5,8,-4),decreasing = TRUE)
```

```
[1] -4 1 3 5 8
```

# Matrix



```
> matrix(seq(from=1,to=31,length=6),ncol=3,byrow=  
  T)
```

```
> matrix(3, 2, 3)
```

```
> matrix(1:6, 2, 3)
```

```
> matrix(1:6, 2, 3, byrow =TRUE)
```

# Matrix computation



- **matrix()**
- **rbind(), cbind()**
- **+, -, \*, /**
- **+, -, %\*%, solve(): solve  $Ax=b$**
- **t(), det(), diag(), qr(),**
- **eigen():**for eigenvalues and eigenvectors
- **svd():**singular value decomposition

# Data Frame and Lists

- List-There is no particular need for the components to be of the mode or type. For example, a list could consist of a numeric vector, a logical value, a matrix, a complex vector, a character array, a function, and so on.
- Dataframe-can be viewed as a matrix with columns possibly of differing modes and attributes

Ex: `> x <- 1:4`

`> y <- 1:3`

`> z <- c("no1", "no2", "no3", "no4")`

`> datFr <- data.frame(x, z)`

`> datFr <- data.frame(N1=x, N2=z)`

`> xyzList <- list(x, y, z)`

`> xyzList <- list(N1=x, N2=y, N3=z)`

# Reading data in a file

- text (ASCII) files
- Excel, SAS, SPSS, ...
- Getting and setting working directory
  - `getwd()`, `setwd()`
- `read.table()`: main way to read data

Ex: `> mydata <- read.table("Bacteria.txt")`

`> mydata <- read.table("c:/data/Bacteria.txt", TRUE)`

`> mydata <- read.table("http://www-stat.stanford.edu/~nzhang/203_web/Data/Bacteria.txt")`

# Reading data in a file

- `scan()`: more flexible than `read.table()`
  - can specify the mode of the variables.  
Ex: `cat("TITLE extra line", "2 3 5 ", "6 8 10",  
file="ex.data", sep="\n")`  
`scan("ex.data", what = list("", "", 0))`
  - can be used to create different objects, such as vectors (default), matrices, data frames, lists, ...
- `read.fwf()`: Read in fixed width format
  - Ex: `read.fwf("test.txt", widths=c(1,2,3))`

# Saving Data

- `write.table(x, file = "", append = FALSE, ...)`
- `write(x, file = "data", sep = " ")`
- `save(x, y, "xy.RData")`
  - `load("xy.RData")`
- `save.image()`
  - `load(".RData")`

# Operators



- Arithmetic

**`+, -, *, /, ^, %%, %/%`**

- Comparison

**`<, <=, >, >=, ==, !=`**

- Logical

**`!, &, &&, |, ||`**

# Control Structures



- if statement

if ( condition ) statement1 else statement2

Ex: if (x>0) y=x else y=0

- for

for (variable in sequence) statement

- while

while ( condition ) statement

- repeat

repeat statement

# The indexing system

- Vector

`x[3]`, `x[1:3]`, `x[c(1, 3)]`, `x[-1]`, `x[-(1:3)]`, `x[-c(1, 3)]`, ...

`x[x >= 5]`, `x[x %% 2 == 0]`, `x[x == 1]`,

`x[c(FALSE, TRUE)]`, ...

- Matrix

`x[2, 3]`, `x[2, ]`, `x[, 3]`, `x[, -(1:2)]`, ...

`x[2, 3, drop = FALSE]`, ...

- Array

`x[3, 2, 4]`, ...

- Ex: `> b=read.table("bacteria.txt", header=T)`

- `> b$t`

- `> b[, "t"]` (or `b[, 1]`)

# names



- Vector

```
names(x) <- c("no1", "no2", "no3")
```

- Matrix

```
rownames(x) <- c("r1", "r2", "r3")
```

```
colnames(x) <- c("c1", "c2", "c3")
```

```
dimnames(x) <- list(c("a", "b"), c("c", "d"), c("e", "f"))
```

- Data Frame

# Graphics

- Two kinds of graphical functions:
  - High-level plotting functions
  - Low-level plotting functions
- High-level graphical functions
  - **plot(x)**
  - **plot(x, y)**
  - **boxplot(x)**
  - **pairs(x)**
  - **hist(x)**
  - **qqplot(x, y)**
  - **image(x, y, z)**
- Low-level plotting command
  - **points(x, y)**
  - **lines(x, y):**
  - **text(x, y, labels, ...)**
  - **segments(x0, y0, x1, y1)**
  - **abline(a, b)**
  - **abline(h=y)**
  - **rect(x1, y1, x2, y2)**
  - **title(str)**

# Graphic Example

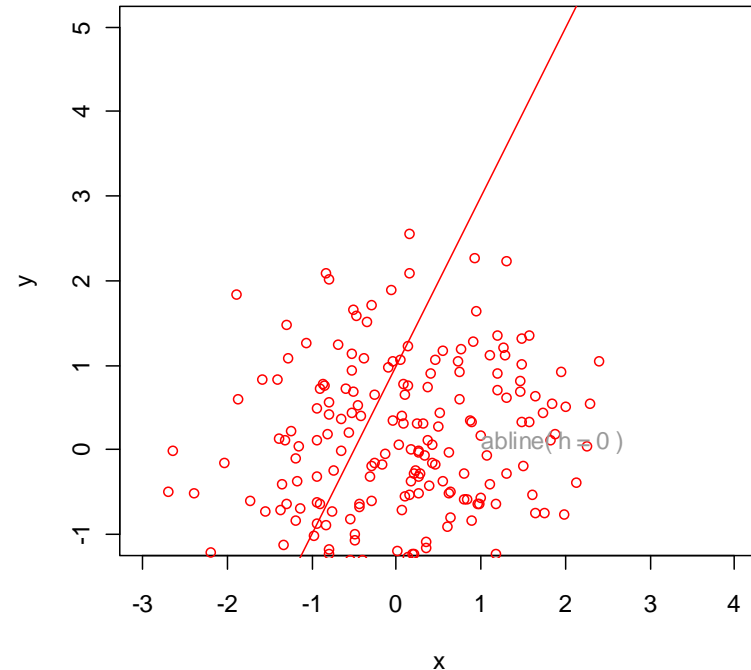
- Ex1 :

```
>plot(c(-2,3), c(-1,5), type  
= "n", xlab="x",  
ylab="y", asp = 1)
```

```
>abline(a=1, b=2, col = 2)
```

```
>points(rnorm(200),  
rnorm(200), col = "red")
```

```
>text(1,0, "abline( h = 0 )",  
col = "gray60", adj =  
c(0, -.1))
```



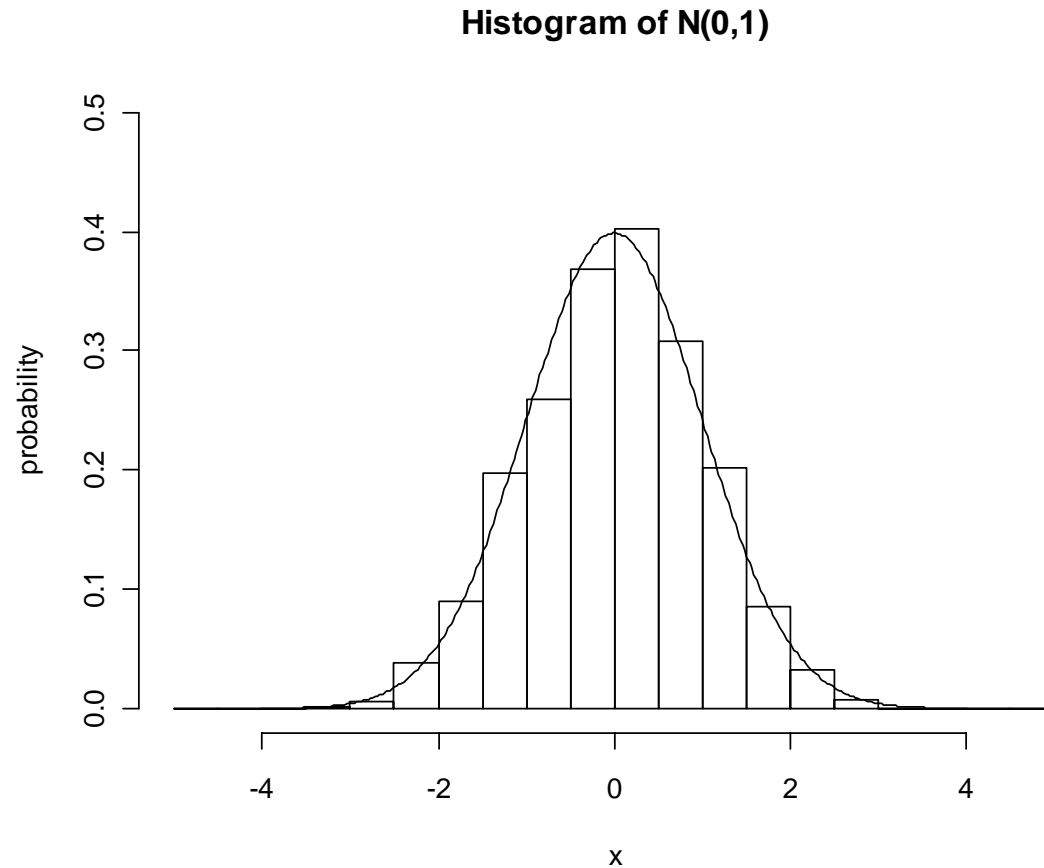
```
>x<-rnorm(1000,0,1)
```

```
>hist(x,ylab="probability", main="Histogram of N(0,1)",
```

```
breaks=seq(from=-5, to=5,length=21), pro=T,ylim=c(0,0.5))
```

```
>y<-seq(from=-4,to=4,length=300)
```

```
>lines(y,dnorm(y,0,1))
```



# Graphical parameters

- some common options
  - type: specifies the type of plot, “p”, “l”,
  - xlim, ylim: specifies the lower and upper limits of the axes.
  - xlab, ylab: annotates the axes
  - main: the main title
  - sub: sub-title (written in a smaller font)
- Par gives us more control to the drawing.
  - > opar <- par(mfrow=c(2,3))
  - > par(...)
  - >plot...
  - > par(opar)

# Writing functions

- open a text editor, such as R's own editor that can be accessed from the File menu and then clicking on New Script.

- Ex:

```
sumab<-function(a, b){  
  s<-a+b  
  print(paste("sum=",s, sep=""))  
}  
  
>sumab(5,3)
```

- Save the script as “.R “ file

## execute the function



```
> source("sumab.R")
```

```
> val <- sumab(8,9)
```

# Statistical Analyses



Example:

```
h=read.table("hotdog.txt", header=T)
```

```
pairs(h)
```

```
h.lm <- lm(cost ~ protein + cont, data = h)
```

```
(or) h.lm <- lm(h$cost ~ h$protein + h$cont)
```

```
> summary(h.lm)
```

```
(or) print(h.lm)
```

# Using the result

---

Example(cont.)

```
> names(h.lm)
```

```
[1] "coefficients" "residuals""effects" "rank"
```

```
[5] "fitted.values" "assign" "qr" "df.residual"
```

```
[9] "xlevels" "call" "terms" "model"
```

```
> h.lm$coefficient
```

# Formulae

- A formula is typically of the form  $y \sim \text{model}$ .
- Common models:
  - $a + b$  linear predictor
  - $a : b$  only the interaction
  - $a * b = a + b + a : b$
  - $(a + b)^n = (a + b)^* \dots^* (a + b)$
  - b: removes the effect of b
  - 1: a regression through the origin ,ex:  $y \sim x - 1$
  - 1: only the intercept
  - l(): to bracket those portions of a model formula where the operators are used in their arithmetic sense, ex:  $y \sim a + l(b+c)$

# Vectorization

- Loops make R slow.
- Example

```
> x <- 1:5000000
```

```
> y <- 1:5000000
```

compare

```
> z <- x + y
```

with

```
> for (i in 1:5000000) z[i] <- x[i]+ y[i]
```

# Vectorization

- using `apply` or `sapply` if the function is not actually recursive

- Ex:

```
> x <- matrix(1:250000, 500, 500)
```

```
> u1 <- vector(length=500)
```

## Compare

```
for(i in 1:500 ){u1 [i]<-mean(x[,i])+2*sd(x[,i])/500 }
```

## With

```
u2 <- apply(x,2, function(y){mean(y)+2*sd(y)/500})
```

# Reference



1. R tutorial given by Jun Li from last year
2. [Rudy Angeles' R tutorial \(2006/07\)](#)
3. <http://cran.r-project.org/>